Python 語言入門

葉平

- 預備知識: 電腦基本知識、進位系統。但不需要學過任何程式語言
- 第一天學完後: 能用 python 快速解決不太複雜的問題
- 第二天學完後: 能用 python 手稿 (script) 和現成模組解決較複雜的問題
- 主要參考: Guide van Rossum and Fred L. Drake Jr., editor, Python Tutorial, release 2.3.2

第一天

- 第一節: 認識 Python
- 第二節: 內建資料型態
- 第三節: 流程控制
- 第四節: 函式
- 第五節: 文件字串、過濾、映射、精煉、載入模組
- 第六節: 撰寫手稿和模組

第一節 認識 Python

1. Python 語言入門
    1. Python 環境的安裝
    2. Python 的應用
    3. Hello, World!
    4. 第二個範例
    5. 第三個範例
    6. 第四個範例
    7. Python 的特色 (Features)
    8. Python 的版本

1 Python 語言入門

葉平

- 預備知識: 電腦基本知識、進位系統。但不需要學過任何程式語言
- 第一天學完後: 能用 python 快速解決不太複雜的問題
- 第二天學完後: 能用 python 手稿 (script) 和現成模組解決較複雜的問題
- 主要參考: Guide van Rossum and Fred L. Drake Jr., editor, Python Tutorial, release 2.3.2

1.1. Python 環境的安裝

Windows: 請助教協助

1.2. Python 的應用

- Anaconda: Red Hat Linux 的安裝程式
- Google 內部的程式
- GNU mailman: mailing list archive
- Zope/Plone: 網路出版環境/知識管理系統
- MoinMoin: Wiki
- pydict: Linux 上的英漢字典
- Online game server

太多了...

1.3. Hello, World!

- 任何語言的第一個範例

```
$ python
Python 2.2 (#3, Sep  1 2002, 20:55:03)
[GCC 2.96 20000731 (Red Hat Linux 7.1 2.96-99)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print 'Hello, world!'
Hello, world!
>>> ^D
$
```

- 解譯式語言 (interpretive)
- 互動式 (interactive)
- 用 End-of-file (Linux: control-D, Windows: control-Z) 結束 python 解譯器

1.4. 第二個範例

- 許多語言的第二個範例: 當簡易計算機用

```
>>> 3+5
8
>>> x = 3**2.5           # 3**2.5 的計算結果用 x 來記住
>>> y = 0
>>> x, y
(15.588457268119896, 0)
>>> x/y
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ZeroDivisionError: float division
```

- 加(+)、減(-)、乘(*)、除(/)、次方(**)、餘(%)
- 變數的指定和使用
- 準確位數 => 單精度還是雙精度?
- 錯誤(error)的顯示
- 註解

1.5. 第三個範例

- 工程計算機

```
>>> from math import *
>>> sqrt(3.0)
1.7320508075688772
>>> log10(2)
0.3010299956639812
>>> sin(pi/3)
0.8660254037844386
```

- math 模組的載入

1.6. 第四個範例

費伯納西數列 (Fibonacci series): A(i+1) = A(i) + A(i-1), A(1) = A(2) = 1

```
>>> a, b = 0, 1
>>> while b < 1000:
...     print b,
...     a, b = b, a+b
...
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

- 多重指定
- 冒號和內縮
- print

1.7. Python 的特色 (Features)

- 解譯式 (interpretive)、互動式 (interactive)
- 模組化 (modular)
- 跨平台 (cross-platform): Linux、Windows、MacIntosh[illegible]許多 UN*Xes...
- 自由 (free software): 自由使用、自由研究、自由修改、自由散佈
- 動態定型 (dynamic typing): 不需要宣告變數型態
- 用內縮表示迴圈和其他的階層
- 內建物件型態 (built-in object types): list、dictionary、string、tuple
- 內建工具 (built-in tools): 切片 (slicing)、連接 (concatenating)、排序 (sorting)、映射 (mapping)、... 等等
- 函式庫: HTML、http、socket、正規表示式(regular expression)、圖形介面(GUI)... 等等
- 第三人工具 (third-party utilities): COM, XML, imaging, ... (自由真好!)
- 非強迫性的物件導向 (object-oriented): 想寫物件就寫物件, 不想寫物件也不會被強迫寫物件。

- 可與其他語言結合:
  - 擴充 (extending): 讓 python 執行你的 C++ code
  - 內嵌 (embedding): 讓 python 成爲你的 C/C++/... 應用軟體的一部分
- 易學、易用

1.8. Python 的版本

- 持續演進的語言
- 新特色持續被提出、討論、同意、實作, 見 Python Enhancement Proposals。

```
1  mylist = [];
2  for i in range(10):
3        mylist.append(i*i)       # old
```

```
1  mylist = [i*i for i in range(10)]              # new
```

- 最新的版本: 2.3.2, 2003/10/3 釋出。
- 本課程內容以 python 2.3 爲主

第二節　內建資料型態

1. 內建物件型態：數字 (Numbers)
2. 內建物件型態：文字 (Text)
3. 內建容器型態：清單 (List)
4. 清單的操作 (List Operations)
5. 清單中有清單 (Lists in a List)
6. 清單的應用 (Using Lists)
7. Tuple
8. 序列 (Sequence)
9. 字典 (Dictionary)

1. 內建物件型態：數字 (Numbers)

- 浮點數: 3.1415, 6.02E23 <64 位元, 雙精度>
- 整數: 100 (32 位元), 0324 <八進位整數>, 0xA140 <十六進位整數>
- 長整數: 5000000000 <任意精度> : 可用來精確表示政府預算 
- 複數: 1-5j <兩個浮點數>
  z = 1-5j, w = 3j+2, z*w = ? w.real, w.imag, w.conjugate(), abs(w)
- 運算符: a = 5.1; b = 3; c = int(a)
  - 加減乘除次方餘 + - * / ** %: 用 a 和 b 全部試一遍, a%b = ?
  - 左括右括 ( ): a / (b-1) = ?
  - 負號 -: -a = ?
  - 整數的位元運算: 補數 ~ 左右移 << >> 和 & 或 | 互斥或 ^
    ~c = ?, c<<b = ?, c>>b = ?, c&b = ?, c|b = ?, c^b = ?
  - 優先順序 (precedence): 括弧 > 負號、補數 > 乘、除、餘 > 加、減 > 左移、右移 > 位元和 & > 位元互斥或 ^ > 位元或 |
    同等優先的算符, 順序是由左而右

2. 內建物件型態：文字 (Text)

以字串(strings)爲資料型態, 字串 = 一串字

- 單引號: 'this is a "single-quoted" string,\n don\'t you like it?'
- 雙引號: "this is a 'double-quoted' string"
- 三引號:

```
'''this is a triply-quoted string
字串中可以換行, 也可以用單引號和雙引號'''
```

字串的處理: s1 = 'Hello', s2 = 'World'

- 連接 (concatenate): s1+', '+s2 -> ?
- 重複 (repeat): '='*40 -> ?
- 索引 (index): s1[3] -> ?
- 循序取出單字 (iteration): for c in s1: print c,
- 單字搜尋 (membership): print 'ell' in s1
- 切片 (slice): s2[1:3] -> ?, s23 -> ?, s2[2:] -> ?, s2[:] -> ?, s2[-3:-1] -> ?, s2[-2:] -> ?
- 長度 (length): len(s1)
- 格式字串: str_student = '班上有 %d 個學生' % n
  其他的格式: %f, %g, %e, %E, %x, %X, %s, ...

3. 內建容器型態：清單 (List)

```
>>> m = [1, 5, 7, -3, -1, 0]
>>> m.append(3)
>>> m
[1, 5, 7, -3, -1, 0, 3]
>>> del m[1:3]
>>> m
[1, -3, -1, 0, 3]
>>> m[-1]
3
```

- 中括號加逗號
- 索引和切片: 和字串類似
- 可變 (mutable)

```
>>> m[1] = -1; m                # 換掉一個項目
[1, -1, -1, 0, 3]
>>> m[2:4] = ['Taipei', 'Tainan', 'Hsin-Chu', 'Taichung']; print m    # 換掉一
些項目
[1, -1, 'Taipei', 'Tainan', 'Hsin-Chu', 'Taichung', 3]
```

4. 清單的操作 (List Operations)

m = [1, 3]

- m 是個物件 (object), 其型態 (type) 是清單
- m 有一些方法 (method) 可以用, 如下
- 添加: m.append('hi') -> [1, 3, 'hi']
- 擴充: m.extend([1,0,-1,-2]) -> [1, 3, 'hi', 1, 0, -1, -2]
- 移除: m.remove('hi') -> [1, 3, 1, 0, -1, -2]
- 插入: m.insert(-1, 3.5) -> [1, 3, 1, 0, -1, 3.5, -2]
- 彈出: m.pop() = -2 -> [1, 3, 1, 0, -1, 3.5]
  m.pop(1) = 3 -> [1, 1, 0, -1, 3.5]
- 計數: m.count(1) -> 2
- 尋找位置: m.index(-1) -> 5, m.index(7) -> ValueError!
- 現場排序: m.sort() -> [-2, -1, 0, 1, 1, 3, 3.5]
- 逆轉: m.reverse() -> [3.5, 3, 1, 1, 0, -1, -2]
- 加、乘、len()?

5. 清單中有清單 (Lists in a List)

```
>>> a = 7
>>> m = [1, 5, a]
>>> a = -1
>>> m
[1, 5, 7]
>>> k = [3, m, 'hello']
>>> k
[3, [1, 5, 7], 'hello']
>>> k[1][2]
7
>>> m[1] = 0
>>> k
[3, [1, 0, 7], 'hello']
```

- 清單中的一般項目: 是複本 (copy)
- 清單中的清單: 是個參考 (reference), 不是複本 (爲什麼？)

6. 清單的應用 (Using Lists)

- 當作堆疊 (stack): <先入後出> 用哪兩個操作進出？
- 當作列隊 (queue): <先入先出> 用哪兩個操作進出？

7. Tuple

- {Webster's dictionary} Tuple: [語源學] set of (so many) elements -- usually used of sets with ordered elements <the ordered 2-tuple (a, b)>. quintuple, sextuple
- 和清單相似, 是一些量的有序集合。
- 和清單最大的不同: 不能變
- 可以作爲 dictionary 的鍵、函式回傳、多變數給值 ... 等等
- 小括號加逗號表示法, 小括弧視情況可省略

```
t = (-1, 3.5, 'hello')
x, y, z = 1, 2, 3
```

8. 序列 (Sequence)

- 有三種: 清單、tuple、字串
- 都有類似的運算: 索引、切片

9. 字典 (Dictionary)

- 鍵 (key) 和值 (value) 的對照表 -->「關連式記憶」

```
>>> prices = { 'apple': 7.5, 'orange': 4.5, 'banana': 2}
>>> prices['orange']                    # 用鍵當索引
4.5
>>> prices.has_key('tomato')
0
>>> prices.keys()                       # note (lack of) order
['orange', 'apple', 'banana']
>>> prices['guava'] = 6.7
>>> print prices['tomato']
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
KeyError: 'tomato'
```

- 鍵必需是不可變的型態 (如: 常數、常字串、Tuple)
- 由 Tuple 建構字典:

```
1  >>> dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
2  {'sape': 4139, 'jack': 4098, 'guido': 4127}
```

第三節　流程控制

1. 流程控制 (Flow Control)
2. 邏輯表示式 (Logic Expression)
3. 迴圈 (Loops)
4. 範圍函式 (The range() Function)
5. 迴圈的分叉
6. 迴圈的小技巧

1. 流程控制 (Flow Control)

```
>>> a = '小華'
>>> if a == '小明':
...     print '[%s]' % a
... elif not a == '小華':
...     print '<%s>' % a
... else:
...     print '(%s)' % a
...
(小華)
```

- if/elif 後的是邏輯表示式 (logic expression)
- 不用把 expression 括起來
- 內縮
- 在同一組邏輯分支中:
  - if 只有一個
  - elif 可以沒有或有多個
  - else 可以沒有或有一個

2. 邏輯表示式 (Logic Expression)

- 表示式的值是僞(False、0、None、空序列即[]、()、) 或真 *(True,非 0 的數,非空序列)*
  - 邏輯非 *not*、和 *and* 、或 *or*
  - 在 *in* 、 不在 *not in*
  - 比較算符: 相等 == 不相等 *!=* 大於 > 小於 < 不小於 >= 不大於 <=
    可串接: 如 $a < b < c != d > g$ 即等同於 $a < b$ *and* $b < c$ *and* $c != d$ *and* $d > g$
- 優先順序: 在、不在 > 比較 > 非 > 和 > 或
- 用括弧括住要先算的部分
- 習題: 以下程式會印出什麼？
  暗示: 檢查部分表示式是否足夠決定整個表示式的真僞 *(「短路算符」)*

```
>>> a, b = 5, 1
>>> def func(x):
...     print 'x = %s in func' % s
...     return 1
>>> a == 5 or func(a)
>>> a > b and func(b)
```

3. 迴圈 (Loops)

```
>>> m = ['apple', 'tangerine', 'banana']
>>> for fruit in m:
...     print fruit, len(fruit)
...     if fruit[0] > 'p':
...         break
...
apple 5
tangerine 9
>>> i = 7
>>> while i > 0:
...     print i**3,
...     i -= 1
...
343 216 125 64 27 8 1
```

- 用不同的變數值反覆執行相同的程式區塊
- *for loop* 語法: *for var in object: loopbody*
- *while loop* 語法: *while expression: loopbody*
- 迴圈內容要內縮 *(indent)*, 用一個跳格或一些空白, 同一階層的要用等量的空白, 空白行結束一階層

4. 範圍函式 *(The range() Function)*

```
>>> range(10)                  # 包括 0, 不包括 10
[0, 1, 2, ... 9]
>>> range(3,6)                 # 包括 3, 不包括 6
[3, 4, 5]
>>> range(2,8,2)               # 每步加 2
[2, 4, 6]
>>> m = [x*x for x in range(4)]    # 清單的內涵建構法 (list comprehension)
>>> for i in range(len(m)):    # 需要索引時
...     print i, m[i]
0 0
1 1
2 4
3 9
```

- 習題: 建構 *[-30, -50, -70, -90]* 的 *range()* 呼叫。 *## >>> range(-30,-100,-20) # 每步加 -20,* 到大於或等於 *-100* 爲止

5. 迴圈的分叉

- *break:* 跳出迴圈
- *continue:* 略過區塊的以下部分、從迴圈的下一次繼續
- *pass:* 無事
- *else:* 在迴圈走到底時執行

```
>>> for n in range(2,10):
...     for x in range(2,n):
...         if n % x == 0:
...             print n, 'equals', x, '*', n/x
...             break
...     else:       # not for "if", but for end-of-for-loop
...         print n, 'is a prime number'
...
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
```

6. 迴圈的小技巧

- 翻遍字典: *items()*

```
>>> for stuff, price in prices.items():
...     print stuff, '的價格是', price
...
orange 的價格是 4.5
apple 的價格是 7.5
banana 的價格是 2
```

- 列出序列的索引: *enumerate() [流水號]*

```
1  >>> m = prices.keys()
2  >>> m.sort()
3  >>> for i, v in enumerate(m):
4  ...     print i, v
5  0 apple
6  1 banana
7  2 orange
```

- 同時 *loop* 兩個序列: *zip()*

```
1  >>> questions = [ name ,  quest ,  favorite color ]
2  >>> answers = [ lancelot ,  the holy grail ,  blue ]
3  >>> for q, a in zip(questions, answers):
4  ...     print  What is your %s? It is %s.  % (q, a)
5  ...
6  What is your name? It is lancelot.
7  What is your quest? It is the holy grail.
8  What is your favorite color? It is blue.
```

第四節　函式

1. 函式 (Functions)
2. 函式：預設引數 (Default Arguments)
3. 函式：關鍵字引數 (Keyword Arguments)
4. 函式： 任意引數 (Arbitrary Arguments)
5. 函式引數的展開
6. 函式引數: 怎麼傳進函式？
7. 匿名函式 (Anonymous Function)
8. 名字 (Names)

1. 函式 (Functions)

```
1   def distance(v0,a,t):
2         return v0*t + a/2.0 * t*t
```

- Function 的中譯: 功能？函數？函式？
- 運算過程所需要的暫存空間: 局部變數 (local variable)
- 傳進函式的引數 (argument): 也視爲局部變數
- return 敘述
- 從外面看：「黑盒子」, 引數進去、結果出來。
- 在呼叫時, 引數有二種:
  - 位置引數 (一般引數): 由引數位置順序決定哪個引數是什麼值。
  - 關鍵字引數: 由引數名字決定
- 習題: 寫 fib(n) 函式產生不大於 n 的費伯納西數列
  fib(100) -> [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

2. 函式：預設引數 (Default Arguments)

預設引數 (default arguments) 提供預設功能

```
>>> def eat(maxprice, lunch='chickenrice', soup='no'):
...       print 'will eat', lunch, 'for lunch with', soup, 'soup for less
than', maxprice
...
>>> eat(150, '天丼', '味噌')       # 都指定了
will eat 天丼 for lunch with 味噌 soup for less than 150
>>> eat(130, '天丼')               # 沒指定湯：用預設的湯
will eat 天丼 for lunch with no soup for less than 130
>>> eat(120)                       # 只指定價錢：用預設的午餐組合
will eat chickenrice for lunch with no soup for less than 120
>>> eat()                          # 未指定一般引數：錯誤!
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: eat() takes at least 1 argument (0 given)
```

- 預設引數值是在定義函式時決定

```
>>> default_soup = 'no'
>>> def eat(maxprice, lunch='chickenrice', soup=default_soup):
...       print 'will eat', lunch, 'for lunch with', soup, 'soup for less
than', maxprice
...
>>> default_soup = '紫菜蛋花'
>>> eat(130)
will eat chickenrice for lunch with no soup
```

- 定義函式時: 有預設的引數必須在位置引數的後方

3. 函式：關鍵字引數 (Keyword Arguments)

```
>>> eat(130, soup='羅宋')                    # 只指定湯, 不指定主食
will eat chickenrice for lunch with 羅宋 soup for less than 130
>>> eat(150, soup='羅宋',lunch='pasta')    # 都指定, 但順序任意
will eat pasta for lunch with 羅宋 soup for less than 150
>>> eat(soup='羅宋',lunch='pasta', 150)    # 錯誤: 位置引數在關鍵字引數之後
SyntaxError: non-keyword arg after keyword arg
```

- 用 關鍵字=值 的方式呼叫
- 順序不限

```
1  >>> def eat(maxprice, lunch='chickenrice', soup='no', **inst):
2  ...     print 'will eat', lunch, 'for lunch with', soup, 'soup for less
   than', maxprice
3  ...     if inst != {}:       # 有特殊指示
4  ...         print 'and the following cooking instructions:'
5  ...         keys = inst.keys();     keys.sort()
6  ...         for kw in keys:     print kw, ':', inst[kw]
7  ...
8  >>> eat(120)
9  will eat chickenrice for lunch with no soup for less than 200
10 >>> eat(200,'beefrice','miso', drink='lemontea', MSG=0 )
11 will eat beefrice for lunch with miso soup for less than 200
12 and the following cooking instructions:
13 MSG : 0
14 drink : lemontea
15 >>> eat(200, drink='lemontea', MSG=0, soup='miso')
16 will eat chickenrice for lunch with miso soup for less than 200
17 and the following cooking instructions:
18 MSG : 0
19 drink : lemontea
```

- **引數: 字典, 必須在最後

- 「不認識」的 (關鍵字,值)對 一律進入 **引數字典
- 彈性, 不限個數

4. 函式： 任意引數 (Arbitrary Arguments)

```
>>> def eat(maxprice, lunch='chickenrice', soup='no', *sides, **inst):
...     print 'will eat', lunch, 'for lunch with', soup, 'soup for less
than', maxprice
...     if sides:
...         print 'plus the following side dishes:',
...         for dish in sides:  print dish,
...         print ''
...     if inst != {}:
...         print 'and the following cooking instructions:'
...         keys = inst.keys();     keys.sort()
...         for kw in keys:     print kw, ':', inst[kw]
...
>>> eat(200)
will eat chickenrice for lunch with no soup for less than 200
>>> eat(200,'beefrice','miso','fish')
will eat beefrice for lunch with miso soup for less than 200
>>> eat(200,'beefrice','miso', drink='lemontea', MSG=0 )
will eat beefrice for lunch with miso soup for less than 200
and the following cooking instructions:
MSG : 0
drink : lemontea
>>> eat(200,'beefrice','miso','fish', salt='less', MSG=0 )
will eat beefrice for lunch with miso soup for less than 200
plus the following side dishes: fish
and the following cooking instructions:
MSG : 0
salt : less
>>> eat(200, drink='lemontea', MSG=0, soup='miso')
will eat chickenrice for lunch with miso soup for less than 200
and the following cooking instructions:
MSG : 0
drink : lemontea
```

- *引數: Tuple, 必須在位置引數之後; 如果有 **引數的後, 必須在其之前
- 彈性, 不限個數

5. 函式引數的展開

```
>>> arglist = [3,6]
>>> range(*arglist)
[3, 4, 5]
```

- 引數現存在一清單或 Tuple 時用

6. 函式引數: 怎麼傳進函式？

- 以參考傳 (pass by object reference): 在函式中看到的引數是外面傳進來的物件的參考
- 如果引數是可變型態, 能改到原物件

```
>>> def double(x):
...     x *= 2
...     return x
...
>>> double(3)
6
>>> a = 5
>>> double(a)
10
>>> a
5
>>> l = [1,3,4]
>>> double(l)
[1, 3, 4, 1, 3, 4]
>>> l                    # mutable, so changed
[1, 3, 4, 1, 3, 4]
```

7. 匿名函式 (Anonymous Function)

```
>>> m = [0., 1., 2., 3.]
>>> def make_incrementor(n):
...     return lambda x: x + n
...
>>> f = make_incrementor(42)      # f 成為一個 +42 的函式
>>> f(0)
42
>>> f(1)
43
```

- 歷史原因, 叫做 "Lambda form" (e.g., LISP)
- 不需要爲了寫一堆小函式而煩惱幫它們一一想名字
- 只能是「一行函式」

8. 名字 (Names)

- 變量 vs 常量
- 局部 vs 廣域

```
>>> favorite = '劉德華'
>>> def friend():
...     favorite = '周杰倫'
...     print favorite
...
>>> print favorite
劉德華
>>> friend()
周杰倫
>>> print favorite
劉德華
>>> del favorite
>>> print favorite
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'favorite' is not defined
```

- 特殊的內建名字: None

第五節　文件字串、過濾、映射、精煉、載入模組

1. 文件字串 (Docstrings)
2. 過濾 (Filtering)
3. 映射 (Mapping)
4. 精煉 (Reducing)
5. 加總 (Sum)
6. 清單內涵 (list comprehension)
7. 載入模組 (Importing Modules)
8. 命名衝突 (Name Clashes)
9. 命名空間 (Namespace)

1. 文件字串 (Docstrings)

```
>>> def eat(arg1, arg2, *args, *inst):
...     '''午餐函式
...
...     這個函式指定我要吃的午餐主食和湯， 可以加點， 也可以給指示 （如： 不要味精）
...     葉平， 某年某月某日'''
...     # function body
...
>>> print eat.__doc__
午餐函式

        這個函式指定我要吃的午餐主食和湯， 可以加點， 也可以給指示 （如： 不要味精）
        葉平， 某年某月某日
```

- 統一的文件方式
- 每個要重複利用的函式都應該包括文件字串

2. 過濾 (Filtering)

```
1  >>> s = 'Hello, World!'
2  >>> def isupper(c):
3  ...     return c in 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
4  >>> filter(isupper,s)
5  'HW'
```

- filtered_sequence = filter(function, sequence)
- 產出的序列和輸入的序列同型態: list -> list, tuple -> typle, string -> string

3. 映射 (Mapping)

```
1  >>> from math import *
2  >>> m = [0., pi/4, pi/2, pi]
3  >>> map(sin, m)
4  [0.0, 0.70710678118654746, 1.0, 1.2246063538223773e-16]
```

- mapped_list = map(function, sequence)
- 習題: scale() 函式開根號乘以十 scale([36., 10., 90.]) -> [60.0, 31.6, 94.9]
  暗示: 函式內的函式
- 可以用 多引數函式 把 多個序列 映射爲一個清單

```
1  >>> def add(x,y):
2  ...     return x+y
3  ...
4  >>> map(add, (-1,-2,-3), (4,3,2))
5  [3, 1, -1]
```

- 可以用 匿名函式

```
1  >>> map(lmabda x,y: x+y, (-1,-2,-3), (4,3,2))
2  [3, 1, -1]
```

4. 精煉 (Reducing)

- 向量長度 = 元素平方和開根號

```
>>> m = [1., 2., 3.]
>>> def vec_length(vec):
...     return math.sqrt(reduce(lambda x,y: x+y, map(lambda x: x*x, vec)))
...
>>> vec_length(m)
3.7416573867739413
```

- 精煉: 把序列精煉出一個值, value = reduce(function, sequence)
- 習題: nearest 函式, 找出序列中離某個值最近的值 nearest([1,4,7,10], 5) -> 4

5. 加總 (Sum)

```
>>> m = [1., 2., 3., 4.]
>>> sum(m)
10.0
```

- 習題: 用精煉作出加總?

6. 清單內涵 (list comprehension)

- 直接指定內涵的形式, 取代迴圈或映射

```
>>> a = [i*i for i in range(5)]
>>> a
[0, 1, 4, 9, 16]
>>> [[x*x, math.sin(x)] for x in a if x < 4]
[[0, 0.0], [1, 0.8414709848078965], [16, -0.7568024953079282]]
>>> vec1 = [2, 4, 6]
>>> vec2 = [4, 3, -9]
>>> [x*y for x in vec1 for y in vec2]
[8, 6, -18, 16, 12, -36, 24, 18, -54]
>>> x, y                                    # 離開清單內涵後仍看得到
(6, -9)
```

- [expression for var in list]
- [expression for var in list for var2 in list]
- [expression for var in list if expression]
- 習題:

7. 載入模組 (Importing Modules)

工程計算機的例子: sin(pi/3)

```
>>> sin(pi/3)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'tan' is not defined
>>> from math import *
>>> sin(pi/3)
0.8660254037844386
```

- 進入 python 環境時並未預載數學模組
- 載入數學模組後就有函式 tan()、 sin()、 sqrt(), ... 和變數 pi、e 可以用

8. 命名衝突 (Name Clashes)

有時候需要載入不只一個模組...

```
>>> from math import *
>>> from project import *
>>> sin(pi/3)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: unsupported operand type(s) for /: 'list' and 'int'
```

- 發生了什麼事？
  - 變數名的衝突 (name clashes)
  - project.py 的內容:

```
pi = '張三教授'
...
```

- 後載入的模組會蓋過先載入的模組...

9. 命名空間 (Namespace)

```
>>> from math import *
>>> import project
>>> sin(pi/3)
0.8660254037844386
>>> project.pi
'\xb1i\xa4T\xb1\xd0\xb1\xc2'
>>> print project.pi
張三教授
```

- OK！
- 點記號法 (dot notation) 可以避免命名衝突: 模組.變數, 模組.函式()
- 避免衝突: 盡量用 import 和點記號法, 少用 from ... import ...
- project.pi 的「亂碼」晚一點講

第六節　撰寫手稿和模組

1. 手稿 (Script)
2. 重複使用手稿
3. 模組和 Byte-Code
4. 影響 Python 的環境變數
5. 模組
6. 套件 (Package)
7. Python 的授權條款

1. 手稿 (Script)

- 常做的運算 -> 存成檔案, 重複使用
- 較長的運算 -> 存成檔案
- Python 程式檔 -> 手稿 (script)

```
$ cat fibo.py
#!/usr/bin/env python
def fibo(n):
        a, b = 0, 1
        while b < n:
                print b, a,
                b = b, a+b
fibo(10)
$ python fibo.py
1 1 2 3 5 8
$ chmod a+x fibo.py
$ ./fibo.py
1 1 2 3 5 8
```

2. 重複使用手稿

```
1  >>> import fibo
2  1 1 2 3 5 8
3  >>>
```

- 問題是什麼？
- `fibo(10)` 在載入時立刻執行

`$ cat fibo.py`

```
1  #!/usr/bin/env python
2  def fibo(n):
3        a, b = 0, 1
4        while b < n:
5              print b, a,
6              b = b, a+b
7  if __name__ == '__main__':
8        fibo(10)
9  $ ./fibo.py        # 獨立執行: okay
10 1 1 2 3 5 8
11 $ python
12    ....
13 >>> import fibo    # 當作模組載入: okay
14 >>> fibo.fibo(20)
15 1 1 2 3 5 8 13
```

- 技巧: name 在手稿被直接執行時值是 '__main__', 在被 import 載入時是模組名 'fibo'
- 注意: 模組檔名一定要以 .py 結束

3. 模組和 Byte-Code

- 載入 -> 解譯 -> 執行
- 效率: 可以不用每次都解譯嗎？
- "Compiled" 模組: spam.py -> 解譯 -> Byte-Code spam.pyc
- 下次載入 spam 時, Python 解譯器會找到 spam.pyc, 如果比 spam.py 新, 就直接載入 spam.pyc 。
- Byte-code 可跨平台: 從 Windows 上 copy 到 Linux, 直接可用, 反之亦然

4. 影響 Python 的環境變數

- `PATH`: 要找得到 python 執行檔
- `PYTHONPATH`: python 用 `PYTHONPATH` 來找模組, 如 `.:/usr/local/lib/python`
- `PYTHONSTARTUP`: 啓動 python 時自動載入的手稿檔名

5. 模組

- 常用模組: sys, os, string
- sys.path 就是 PYTHONPATH
- 模組查詢: dir()
- 習題: 載入 fibo 模組, 請解釋 dir(fibo) 看到的是什麼。

6. 套件 (Package)

- 有強關連性的模組組成一個套件 (package)
- 幾個套件可以組成更大的套件
- 例: python 的 xml 套件。
  - xml.sax, xml.dom, xml.parsers 三個子套件

```
1 import xml.sax.xmlreader
    # 引入 xml 套件的 sax 子套件的 xmlreader 模組
2 loc = xml.sax.xmlreader.Locator()    # 建立一個 Locator 物件
3 from xml.dom import minidom
    # 引入 xml 套件的 dom 子套件的 minidom 模組
4 minidom.parse('test.xml')
5 from xml.dom.pulldom import parse
    # 引入 xml 套件的 dom 子套件的 pulldom 模組的 parse 函式
6 a = parse('an XML string')
```

- import 的寫法和函式的用法息息相關
- 一個模組是一個 .py 檔 (sys 模組在 $PYTHONPATH 中的 sys.py), 一個套件是一個目錄，內含其模組的 .py 檔, 以及必要的輔助檔。

7. Python 的授權條款

- 細節: http://www.python.org/psf/license.html
- 版權: 絕大部分的 python 程式爲 Python 軟體基金會所有, 但整個 python 程式皆受 Python License 規範。
- 主要特色:
  - 自由使用: 商業或非商業皆可, 內嵌或擴充都不用付授權費。
  - 自由散佈: 用源碼或 binary 方式散佈原版或修改過的 python 都可以, 也可以用 binary 方式散佈你的 python 擴充模組。
  - 在你散佈的版本中, 不可以移除 Python 軟體基金會的版權宣告。
  - 開放源碼創進會 (Open Source Initiative) 已認證 Python License 爲開放源碼的授權條款
  - 自由軟體基金會也認定 Python License 和 GPL 相容。

第二天

1. 基本檔案操作 (Basic File Operations)
2. 影響 Python 的環境變數
3. 例外 (Exceptions)
4. 例外處理 (Exception Handling)
5. 產生例外狀況
6. 參考、指定 (References, Assignments)
7. 參考和收垃圾 (References and Garbage Collection)
8. 眼界 (Scopes)
9. 樣式比對 (Pattern Matching)
10. 樣式比對-II (Pattern Matching)
11. 正規表示式 (Regular Expressions)
12. 中文處理
13. 字串物件 (string object)
14. Python 敘述
15. 程式設計 (Programming)
16. 物件導向程式設計 (Object Oriented Programming)
17. 類別 (Classes)
18. 自訂使用方法
19. 繼承 (Inheritance)
20. 資料一致性 (Data Consistency)
21. 算符重載 (Operator Overload)
22. 循序器 (Iterators)

1. 基本檔案操作 (Basic File Operations)

```
>>> inp = open('hello.txt', 'r')    # 開啟 hello.txt, 唯讀。 inp 是檔案物件
>>> for line in inp.readlines():    # readlines() 得到清單, 每一項目是一行
...     print line[0]
>>> inp.read()                      # 已到檔尾, 再讀.. 就沒有了
''
>>> inp.close()                     # 關閉
```

```
>>> inp = open('hello.txt', 'r')    # 開啟 hello.txt, 唯讀。 inp 是檔案物件
>>> line = inp.readline()           # 讀入一行, 得到的 line[-1] 是 '\n'
>>> the_rest = inp.read()           # 讀入所有剩下的內容
>>> print the_rest                  # 印出檔案的全部內容
>>> inp.close()                     # 關閉
>>> out = open('world.txt', 'w')    # 開啟 world.txt, 唯寫
>>> out.write('hello, world!\n')    # 寫字
>>> out.close()
```

- open(): 開啓檔案 open(name, mode), 結果: 檔案物件 (file object)
  - mode: 'r' 唯讀, 'w' 唯寫 (洗掉同名檔案)、 'a' 在檔尾添加、 'r+' 又讀又寫
  - mode 可不指定, 與 'r' 同
  - 在 Windows 平台可加指定 'b' 表示是 binary file, 如 'r+b'、 'wb'
- 讀字: read(n), 最多讀 n 個字進記憶體, 不指定 n 則讀進整個檔

2. 影響 Python 的環境變數

- PATH: 要找得到 python 執行檔
- PYTHONPATH: python 用 PYTHONPATH 來找模組
- PYTHONSTARTUP: 啓動 python 時自動載入的手稿檔名

3. 例外 (Exceptions)

- 許多運算一般而言是正確的, 但會有例外情形

```
>>> def myfunc(a, b):
...     return math.log(a/b)
>>> myfunc(2,0)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "<stdin>", line 2, in myfunc
ZeroDivisionError: integer division or modulo by zero
>>> myfunc(2,-1)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "<stdin>", line 2, in myfunc
ValueError: math domain error
>>>
```

- 例外發生時, python 解譯器會印出錯誤訊息和回溯訊息, 然後停下來

- 不想 python 解譯器或手稿停止執行 -> 要處理例外情形

4. 例外處理 (Exception Handling)

```
1  >>> try:                                  # 試跑看看
2  ...     f = file('hello.txt', 'r')
3  ... except IOError, message:             # 如果有 IOError 這個例外
4  ...     sys.stderr.write("%s\n" % message)
5  ... else:                                # 不然
6  ...     content = f.read()
7  ...     f.close()
8  ...
9  [Errno 2] No such file or directory: 'hello.txt'
```

- 如果發生了未列入處理的例外, 就會傳到「上一層」去處理

5. 產生例外狀況

- 比方說: 輸入不符期望時

```
>>> def havefun(x):
...     try:
...         if len(x) > 2:     # 太多東西就 no fun
...             raise 'Too many work to do!'     # 產生例外
...         print x, 'is okay'
...     except TypeError:      # x 不可數
...         print x, 'is fun'
...     return
...
>>> havefun(7.5)
7.5 is fun
>>> havefun([5, 2])
[5, 2] is okay
>>> havefun('abc')
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "<stdin>", line 4, in havefun
Too many work to do!
```

6. 參考、指定 (References, Assignments)

- 第一次指定: x = 'abc'
  在命名空間中建立名字 x, 存放對物件 'abc' 的參考, 物件 'abc' 的參考計數 (reference count) = 1
- 物件 'abc' 是本尊, x 是分身, 本尊的參考計數就是分身個數
- 重新指定: x = 79
  名字 x 成爲物件 79 的分身, 本尊物件 'abc' 的參考計數減一

```
1  x = 9          # x 存有物件『常數 9』的參考, 『常數 9』的參考計數 = 1
2  x = 5          # x 存有物件『常數 5』的參考, 『常數 9』的參考計數減一
```

```
1  >>> L = [x, 2, 3]     # L 存有清單[1,2,3]的參考, 清單項目 0 存有常數 5 的參考, 等等
2  >>> x = 7             # x 存有常數 7 的參考, L[0] 仍存有常數 5 的參考
3  >>> L
4  [5, 2, 3]
5  >>> M = L             # M 存有和 L 一樣的參考
6  >>> T = ('hi', L, 'ho') # T 存有 Tuple 的參考, Tuple 項目 1 存有清單[5,0,3]的參考,
7                       # 清單[1,2,3]的參考計數為 2
8  >>> L[1] = 0          # L[1]存有常數 0 的參考, 常數 2 的參考計數減一
9  >>> M, T
10 ([5, 0, 3], ('hi', [5, 0, 3], 'ho'))
```

7. 參考和收垃圾 (References and Garbage Collection)

```
1  >>> L
2  [5, 0, 3]
3  >>> id(L), id(M), id(T[1])     # id(l) 可視為 L 所存的參考
4  (1074489772, 1074489772, 1074489772)
5  >>> K = [5, 0, 3]
6  >>> K == L
7  True
8  >>> K is L
9  False
10 >>> M is L
11 True
```

- a == b 兩物件相等, a is b 兩物件有同一個本尊。
- 任何物件在參考計數爲 0 後, python 解譯器就可以回收其所使用的記憶體, 稱爲「收垃圾」(garbage collection)。
- 收垃圾的時機由 python 解譯器自行決定

8. 眼界 (Scopes)

- 眼界 = 可直接取用的命名空間
- 有三層 (Nested Scopes):
  - 內層: 函式或類別的命名空間, 「局部名字」, 最先搜尋
  - 中層: 本模組, 「廣域名字」
  - 外層: Python 的內建空間, 「內建名字」, 最後搜尋

搜尋名字的順序又稱爲 LGB 法則

- global 宣告

```
 1  a = 3
 2  def oh():
 3      print a
 4  def hmmm():
 5      a = -3
 6      print a
 7  def wow():
 8      global a
 9      a = -7
10      print a
11  def uhoh():
12      print a
13      a= -3
14      print a
15  print a;  oh();    print a        # 3, 3, 3: 都指到同一個 a
16  print a;  hmmm();  print a        # 3, -3, 3: 中間的是 hmmm() 的局部變數
17  print a;  wow();   print a        # 3, -7, -7: 改到廣域變數
18  print a;  uhoh();  print a        # UnboundLocalError: local variable 'a'
    referenced before assignment
```

- 中層的變數是唯讀的 (read-only), 不能改變。用 global 宣告之後, 才能改變。
- 外層的變數: 猜猜看, 可以改變嗎？

9. 樣式比對 (Pattern Matching)

- 正規表示式: 強大的樣式比對函式庫
- 在 Python 中的實作: re 模組
- 搜尋、取代、分割
- 搜尋: 在字串中尋找符合樣式的部分, 找到: 傳回 Match 類別的物件
  - re.match(pattern, string, flags): 只在字串的開頭中尋找
  - re.search(pattern, string, flags): 傳回第一個符合
  - re.findall(pattern, string): 傳回所有的符合字串
  - re.finditer(pattern, string): 傳回一個循序器 (iterator), 可用來抓出所有的符合物件

```
>>> import re
>>> a = 'Some needs protection, some protects others, and some are well protected.'
>>> match = re.search('protection|protected|protect', a)     # 三者之一
>>> match.groups(), match.start(), match.end(), match.group(0)
((), 11, 21, 'protection')
>>> match = re.findall('protection|protected|protect', a)
>>> match
['protection', 'protect', 'protected']
>>> match = re.finditer('protection|protected|protect', a)
>>> for m in match:
...     print m.start(), m.end(), a[m.start():m.end()]
...
11 21 protection
28 35 protect
63 72 protected
```

10. 樣式比對-II (Pattern Matching)

- 重覆使用同一個樣式: 用 compile 提升效率: pattern = re.compile(patstr,flag)
- 取代: 把找到的字串換掉

```
>>> pat = re.compile('protection|protected|protect')
>>> pat.sub('food', a)                                    # 好笑的句子  :D
'Some needs food, some foods others, and some are well food.'
>>> def repl(match):
...     dict = {'protection': 'food', 'protect': 'feed', 'protected': 'fed'}
...     return dict[match.group(0)]
...
>>> pat.sub(repl, a)
'Some needs food, some feeds others, and some are well fed.'
```

- 分割: 用找到的字串將整個字串分割成一個子字串清單

```
>>> pat.split(a)
['Some needs ', 'protection', ', some ', 'protect', 's others, and some are well ', 'protected', '.']
```

11. 正規表示式 (Regular Expressions)

- 用來表示樣式, 參考 Python Library References 的相關章節
- 一些常用的樣式:

.	任意字元 (除非用了 DOTALL flag, 不然換行字元不算)
[xyz]	字元 x 或 y 或 z
^ $	首和尾 (如果用了 MULTILINE flag 就是行首和行尾)
* + ?	重複前一個樣式 0..N, 1..N, 0..1 次, 加一個 ? 關掉貪婪比對
{m} {m,n}	重複前一個樣式 m 或 m 到 n 次
A | B	樣式 A 或樣式 B
(...)	成組, 之後用 \1 \2 來指定第一組、第二組...
(?P<name>...)	爲組命名, 之後用 group('name') 來取用找到的字串

12. 中文處理

- Unicode 物件: u'string in UCS'
- Big5 字串:

```
>>> big5str = '這是 Big5 中文字串'
>>> ucsstr = unicode(big5str, 'big5-2003')
>>> ucsstr
u'\u9019\u662f Big5 \u4e2d\u6587\u5b57\u4e32'
>>> print ucsstr.encode('utf-8')
'\xe9\x80\x99\xe6\x98\xaf Big5
\xe4\xb8\xad\xe6\x96\x87\xe5\xad\x97\xe4\xb8\xb2'
>>> print ucsstr.encode('big5-2003')
這是 Big5 中文字串
```

- 「許功蓋」問題: 注意反斜線

13. 字串物件 (string object)

s = ' Hello, World! This is a python string. \n\n'

- s.strip() -> 'Hello, World! This is a python string.'
- s.split() -> ['Hello,', 'World!', 'This', 'is', 'a', 'python', 'string.']
- atoi(), atof()
- s.find('a') -> 25, s.find('P') -> -1
- s.index('a') -> 25, s.index('P') -> ValueError: substring not found

14. Python 敘述

- 正常: 一行一個敘述
- 以內縮表示區塊的階層
- 一行多個敘述: 敘述 1; 敘述 2; 敘述 3
- 跨行敘述: 反斜線, 開口的括弧

```
if a == 1 and b == 2 and \
   c == 3:
a_variable = function_a(parameter_1, parameter_2,
                parameter_3, parameter_4)
```

15. 程式設計 (Programming)

- 電腦語言的進展: 機器碼、組合語言、高階語言
- BASIC 時代: 整個程式從頭到尾就一支
- 程序式設計 (procedural): Fortran, C, pascal, ...
  - 把程式分割成函式, 每個函式提供特定的功能
  - 比方說: vadd(v1, v2), vinprod(v1, v2) 把兩個陣列當向量做相加或內積, 資料結構是陣列
  - 資料結構 (data structure): 把相關的資料綁在一起, 如: 電話簿的一筆資料包括姓名(字串)、電話(數字或字串)、地址、...等等
- 物件導向式設計 (oo): Lisp, C++, Java, python, ...

16. 物件導向程式設計 (Object Oriented Programming)

- 資料 (data) 和 對此資料能做的事 (operation) 事先規劃好, 建立成資料型態 (type) 或類別 (class)
  - v1 和 v2 都是 vector class 的物件, 則可以做 v1+v2, v1*v2, v1.len() -> 直觀
- 類別會越來越多, 之間開始發生關係:
  - 有的類別的物件會擁有 (own) 其他類別的物件 [桌子有 4 隻腳]
  - 有的類別是別的類別的特殊情形 (is-a) [電腦桌是一種桌子]
  - 有的類和其他類別相關 (is associated with) [小明是小華的朋友]
- 和日常生活中的經驗比較接近

17. 類別 (Classes)

```
class Table:
    '''The class for tables'''
    def __init__(self, name, color='白'):
        self.name = name
        self.color = color
        self.nlegs = 4
        self.attitude = '朝上'
a = Table('電腦室#1')
print a
<__main__.Table instance at 0x401c208c>
```

- Table 是類別, a 是物件, name、color、nlegs、attitude 是屬性 (attributes)
- docstring 是個好習慣
- init: 建造器 (constructor)
- self: 物件本身的參考, self.name: 物件自己的資料
- 加一個使用方法 (method) 好列印物件資料

```
    def __repr__(self):
        return '%s 放的%s 色的桌子%s' % (self.attitude, self.color,
self.name)

a = Table('電腦室#2','灰')
print a
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□#2
```

- print 或 '%s' 會使用 repr() 來產生物件的字串呈現 (representation)

18. 自訂使用方法

```
class Table:
    ...
    def flip(self):
        if self.attitude == '朝上': self.attitude = '朝下'
        elif self.attitude == '朝下': self.attitude = '朝上'

a = Table('電腦室#2','灰')
a.flip()
print a
朝下放的灰色的桌子電腦室#2
```

19. 繼承 (Inheritance)

```
1  class ComputerTable(Table):
2      '''A computer table'''
3      def __init__(self,name):
4          Table.__init__(self,name,'灰')
5          self.capacity = 2
6          self.computer = []
7
8  c = ComputerTable('#3')
9  c.computer = ['3001', '3002']
```

- 呼叫 base class 的 init
- 增加自己獨有的屬性
- 萬一有人寫 c.computer = ['3001', '3002', '3003'] ?

```
1      def addComputer(self,comp):
2          if len(self.computer) == self.capacity:     # 滿載
3              sys.stderr.write('電腦桌 %s 已經放滿了!\n' % self.name) # 或 raise
4          self.computer.append(comp)
5  c.addComputer('3001')
6  c.addComputer('3002')
7  c.addComputer('3003')
8  c.flip()      # 大災難!
```

20. 資料一致性 (Data Consistency)

以及... 資料隱藏 (Data Encapsulation)

- 提供對資料的取用方法 (getters) 和改變方法 (setters), 以達成資料的一致性

```
1      def removeComputer(self, comp):
2          if comp in self.computer:
3              self.computer.remove(comp)
4          else:
5              sys.stderr.write('電腦桌 %s 上沒有 %s 這台電腦!\n' % (self.name,
  comp))
```

21. 算符重載 (Operator Overload)

```
    def flip(self):
        if self.computer == []:
            __Table__.flip()
        else:       # 避免摔電腦
            sys.stderr.write('%s上有電腦%d部, 不可以 flip!\n' % (self.name, len
(self.computer)) )
            return -1
 c.flip()     # OK, 不會改變姿勢
```

22. 循序器 (Iterators)

```
1   >>> for var in object:
2   >>>     do something ...
```

- Python 2.3 加入的功能
- in 算符: 呼叫 object 的循序器 (iterator), 把結果填到 var 中, 再執行迴圈內容
- 清單的循序器